Introduction to GeoPandas

Python Environment Setup (5 minutes)

- Verify installation and dependencies:
  - Check that students have GeoPandas, Matplotlib, and Jupyter installed
  - Troubleshoot common installation issues
  - Launch Jupyter notebooks
- Introduce the working environment:
  - Explain notebook interface for beginners
  - Set working directory with relevant data
  - Import necessary libraries

GeoPandas Basics: Reading/Writing Spatial Data (8 minutes)

- Loading various formats:
  - Shapefiles, GeoJSON, GeoPackage
  - Introduction to the GeoDataFrame object
  - Understanding the geometry column
- Basic geometric properties and attributes:
  - Accessing coordinates
  - Getting geometric properties (area, length, etc.)
  - Coordinate Reference Systems (CRS) handling
- Saving outputs:
  - Writing to different spatial formats
  - Considerations for preserving attribute data

Data Manipulation and Filtering (7 minutes)

- Pandas-like operations on spatial data:
  - Selecting columns and rows
  - Creating new attributes
  - Grouping and aggregating by attributes
- Spatial filtering:
  - Selection by location/proximity

    - Using spatial predicates (within, contains, intersects)
    - Combining attribute and spatial queries

Simple Spatial Operations (7 minutes)

- Geometric operations:
    - Buffering
    - Spatial joins
    - Dissolving/merging geometries
    - Clipping and intersection analysis
- Raster integration:
    - Basic introduction to rasterio with GeoPandas
    - Extracting values from rasters to points/polygons

Hands-on Exercise: Basic Data Manipulation with GeoPandas (3 minutes setup + students work)

- Provide step-by-step instructions for students to follow
- Include checkpoints where they should verify results

Python Examples

```
# Basic import and loading data
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np

# Load data
city_boundaries = gpd.read_file("data/city_boundaries.shp")
points_of_interest = gpd.read_file("data/points_of_interest.geojson")

# Check CRS and reproject if needed
print(f"City boundaries CRS: {city_boundaries.crs}")
print(f"POIs CRS: {points_of_interest.crs}")

if city_boundaries.crs != points_of_interest.crs:
    points_of_interest = points_of_interest.to_crs(city_boundaries.crs)

# Basic visualization
fig, ax = plt.subplots(figsize=(10, 10))
city_boundaries.plot(ax=ax, color='lightgrey', edgecolor='black')
points_of_interest.plot(ax=ax, color='red', markersize=5)
```

```
plt.title("City Boundaries with Points of Interest")
plt.show()
```

```
# Creating a grid over a study area
from shapely.geometry import box

# Get the total bounds of the city
xmin, ymin, xmax, ymax = city_boundaries.total_bounds

# Define cell size in the CRS units (e.g., 1000 meters)
cell_size = 1000

# Calculate number of cells
nx = int((xmax - xmin) / cell_size)
ny = int((ymax - ymin) / cell_size)

# Create the grid cells
grid_cells = []
for y in range(ny):
    for x in range(nx):
        x0 = xmin + x * cell_size
        y0 = ymin + y * cell_size
        x1 = x0 + cell_size
        y1 = y0 + cell_size
        grid_cells.append(box(x0, y0, x1, y1))

# Create a GeoDataFrame from the grid
grid = gpd.GeoDataFrame({'geometry': grid_cells}, crs=city_boundaries.crs)

# Clip the grid to the city boundaries
grid = grid[grid.intersects(city_boundaries.unary_union)]

# Plot the grid with city boundaries
fig, ax = plt.subplots(figsize=(10, 10))
city_boundaries.plot(ax=ax, color='lightgrey', edgecolor='black')
grid.plot(ax=ax, facecolor='none', edgecolor='blue', alpha=0.5)
plt.title("Regular Grid over City")
plt.show()
```

```
# Spatial join to count points in each grid cell
joined = gpd.sjoin(grid, points_of_interest, how="left",
predicate="contains")
point_counts =
joined.groupby(joined.index).size().reset_index(name='point_count')
```

```
grid = grid.merge(point_counts, left_index=True, right_on='index',
how='left')
grid['point_count'] = grid['point_count'].fillna(0)

# Visualize the point density
fig, ax = plt.subplots(figsize=(12, 10))
grid.plot(column='point_count', ax=ax, cmap='YlOrRd',
          edgecolor='grey', alpha=0.7, legend=True,
          legend_kwds={'label': "Points per Grid Cell"})
plt.title("Point Density by Grid Cell")
plt.show()
```

```
# Advanced analysis: Calculating accessibility metrics
# Buffer analysis to find POIs within 500m of each cell centroid

# Calculate centroids of grid cells
grid['centroid'] = grid.geometry.centroid

# Create a new GeoDataFrame with just the centroids
centroids = gpd.GeoDataFrame(grid[['point_count']],
                             geometry=grid.centroid,
                             crs=grid.crs)

# Buffer the centroids
buffer_distance = 500  # meters
centroids['buffer'] = centroids.geometry.buffer(buffer_distance)

# Create a new GeoDataFrame with the buffer geometries
buffers = gpd.GeoDataFrame(centroids[['point_count']],
                           geometry=centroids.buffer,
                           crs=centroids.crs)

# Count POIs within each buffer
joined_buffers = gpd.sjoin(buffers, points_of_interest, how="left",
predicate="contains")
poi_in_buffer =
joined_buffers.groupby(joined_buffers.index).size().reset_index(name='poi_wi
thin_500m')
grid = grid.merge(poi_in_buffer, left_index=True, right_on='index',
how='left')
grid['poi_within_500m'] = grid['poi_within_500m'].fillna(0)

# Calculate accessibility ratio
grid['accessibility_score'] = grid['poi_within_500m'] /
grid['poi_within_500m'].max()
```

```
# Visualize accessibility
fig, ax = plt.subplots(figsize=(12, 10))
grid.plot(column='accessibility_score', ax=ax, cmap='viridis',
          edgecolor='grey', alpha=0.7, legend=True,
          legend_kwds={'label': "Accessibility Score"})
plt.title("POI Accessibility Score by Grid Cell")
plt.show()
```

```
# Comparing grid analysis with administrative boundaries
neighborhoods = gpd.read_file("data/neighborhoods.shp")

# Spatial join to count points in each neighborhood
joined_hoods = gpd.sjoin(neighborhoods, points_of_interest, how="left",
predicate="contains")
hood_counts =
joined_hoods.groupby(joined_hoods.index).size().reset_index(name='point_coun
t')
neighborhoods = neighborhoods.merge(hood_counts, left_index=True,
right_on='index', how='left')
neighborhoods['point_count'] = neighborhoods['point_count'].fillna(0)

# Calculate point density for meaningful comparison
neighborhoods['area_km2'] = neighborhoods.geometry.area / 1000000  # Convert
to km²
neighborhoods['point_density'] = neighborhoods['point_count'] /
neighborhoods['area_km2']

# Create side-by-side comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

# Grid-based view
grid.plot(column='point_count', ax=ax1, cmap='YlOrRd',
          edgecolor='grey', alpha=0.7, legend=True,
          legend_kwds={'label': "Points per Grid Cell"})
ax1.set_title("Point Distribution (Grid Analysis)")

# Administrative boundary view
neighborhoods.plot(column='point_density', ax=ax2, cmap='YlOrRd',
                   edgecolor='black', alpha=0.7, legend=True,
                   legend_kwds={'label': "Points per km²"})
ax2.set_title("Point Distribution (Neighborhood Analysis)")

plt.tight_layout()
plt.show()
```

```
# Save the results for use in QGIS
grid.to_file("outputs/analysis_grid.gpkg", driver="GPKG")
neighborhoods.to_file("outputs/analysis_neighborhoods.gpkg", driver="GPKG")
```

GeoPandas Hands-On Exercise: Grid-Based Analysis Workflow

In this exercise, you'll learn to create a grid-based analysis of point data (such as amenities, incidents, or facilities) using GeoPandas. This approach helps overcome the biases inherent in administrative boundaries.

Exercise Setup

Required Data

- A polygon boundary file (e.g., city limits, study area)
- A point dataset (e.g., service locations, incidents, features of interest)
- Optional: Administrative boundaries (e.g., neighborhoods, census tracts)

Steps Overview

1. Import libraries and load data
2. Create a regular grid over the study area
3. Analyze point distribution using the grid
4. Compare with administrative boundaries
5. Export results for visualization in QGIS

Detailed Step-by-Step Instructions

Step 1: Set up environment and load data

```
# Import necessary libraries
import geopandas as gpd
import matplotlib.pyplot as plt
import numpy as np
from shapely.geometry import box

# Load your spatial data
boundary = gpd.read_file("study_area.shp")  # Your city or study area
boundary
points = gpd.read_file("points.shp")        # Your points of interest
```

```
admin = gpd.read_file("admin_areas.shp")    # Administrative boundaries
(optional)

# Ensure all data is in the same CRS
points = points.to_crs(boundary.crs)
if 'admin' in locals():
    admin = admin.to_crs(boundary.crs)

# Quick visualization to check data
fig, ax = plt.subplots(figsize=(10, 8))
boundary.plot(ax=ax, color='lightgrey', edgecolor='black')
points.plot(ax=ax, color='red', markersize=5)
plt.title("Study Area with Points of Interest")
plt.show()
```

Step 2: Create a regular grid

```
# Get the bounding box of your study area
xmin, ymin, xmax, ymax = boundary.total_bounds

# Define your grid cell size (in CRS units, e.g., meters)
cell_size = 1000  # Adjust based on your study area size

# Calculate number of cells in each direction
nx = int((xmax - xmin) / cell_size)
ny = int((ymax - ymin) / cell_size)

# Create grid cells
grid_cells = []
for y in range(ny):
    for x in range(nx):
        x0 = xmin + x * cell_size
        y0 = ymin + y * cell_size
        x1 = x0 + cell_size
        y1 = y0 + cell_size
        grid_cells.append(box(x0, y0, x1, y1))

# Create a GeoDataFrame from the grid
grid = gpd.GeoDataFrame({'geometry': grid_cells, 'cell_id':
range(len(grid_cells))},
                        crs=boundary.crs)

# Clip the grid to your study area boundary
grid = grid[grid.intersects(boundary.unary_union)]
```

```
# Visualize the grid
fig, ax = plt.subplots(figsize=(10, 8))
boundary.plot(ax=ax, color='lightgrey', edgecolor='black')
grid.plot(ax=ax, facecolor='none', edgecolor='blue', alpha=0.5)
plt.title("Regular Grid over Study Area")
plt.show()
```

Step 3: Perform point-in-polygon analysis with the grid

```
# Count points in each grid cell using spatial join
joined = gpd.sjoin(grid, points, how="left", predicate="contains")
counts = joined.groupby('cell_id').size().reset_index(name='point_count')

# Join counts back to the grid
grid = grid.merge(counts, on='cell_id', how='left')
grid['point_count'] = grid['point_count'].fillna(0)

# Calculate point density (points per km²)
grid['area_km2'] = grid.geometry.area / 1000000  # Convert to km²
grid['point_density'] = grid['point_count'] / grid['area_km2']

# Visualize the results
fig, ax = plt.subplots(figsize=(12, 10))
grid.plot(column='point_density', ax=ax, cmap='YlOrRd',
          edgecolor='grey', alpha=0.7, legend=True,
          legend_kwds={'label': "Points per km²"})
plt.title("Point Density by Grid Cell")
plt.show()
```

Step 4: Compare with administrative boundaries (optional)

```
# If using administrative boundaries, perform the same analysis
joined_admin = gpd.sjoin(admin, points, how="left", predicate="contains")
admin_counts =
joined_admin.groupby('NAME').size().reset_index(name='point_count')

# Join counts back to admin areas
admin = admin.merge(admin_counts, on='NAME', how='left')
admin['point_count'] = admin['point_count'].fillna(0)

# Calculate point density for admin areas
admin['area_km2'] = admin.geometry.area / 1000000
admin['point_density'] = admin['point_count'] / admin['area_km2']
```

```
# Create side-by-side comparison
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))

# Grid-based analysis
grid.plot(column='point_density', ax=ax1, cmap='YlOrRd',
          edgecolor='grey', alpha=0.7, legend=True,
          legend_kwds={'label': "Points per km²"})
ax1.set_title("Point Density (Grid Analysis)")

# Administrative boundary analysis
admin.plot(column='point_density', ax=ax2, cmap='YlOrRd',
           edgecolor='black', alpha=0.7, legend=True,
           legend_kwds={'label': "Points per km²"})
ax2.set_title("Point Density (Administrative Areas)")

plt.tight_layout()
plt.show()
```

Step 5: Advanced analysis - Hot spot identification

```
# Identify grid cells with significantly high point density
mean_density = grid['point_density'].mean()
std_density = grid['point_density'].std()

# Flag hot spots (> 2 standard deviations above mean)
grid['is_hotspot'] = grid['point_density'] > (mean_density + 2 *
std_density)

# Visualize hotspots
fig, ax = plt.subplots(figsize=(12, 10))
grid.plot(ax=ax, color='lightgrey', edgecolor='grey')
grid[grid['is_hotspot']].plot(ax=ax, color='red', edgecolor='black')
points.plot(ax=ax, color='blue', markersize=3, alpha=0.5)
plt.title("Identified Hot Spots")
plt.show()
```

Step 6: Export results to continue in QGIS

```
# Save the grid analysis result for use in QGIS
grid.to_file("grid_analysis.gpkg", driver="GPKG")

# Optional: Save hot spots as a separate file
grid[grid['is_hotspot']].to_file("hotspots.gpkg", driver="GPKG")
```

Challenge Exercises

1. Create a hexagonal grid instead of a rectangular grid
2. Implement a kernel density estimation over the study area
3. Calculate the distance from each grid cell centroid to the nearest point
4. Identify clusters using DBSCAN from scikit-learn with spatial coordinates
5. Create a choropleth map showing the difference between grid-based and admin-based density measures

Tips for Effective Grid Analysis

1. **Choosing an appropriate cell size:**
   - Too large: misses local patterns
   - Too small: creates sparse data and statistical noise
   - A good rule of thumb: aim for 10-30 cells across your study area
2. **Dealing with edge effects:**
   - Consider clipping cells that are less than 50% within the study area
   - Or weight analysis by the percentage of each cell within the boundary
3. **Alternative grid types:**

```
# For hexagonal grids, use the h3 library
# pip install h3
import h3
# See documentation for implementation details
```

4. **Visualizing in 3D:**

```
from mpl_toolkits.mplot3d import Axes3D

# Extract centroids for 3D plotting
grid['x'] = grid.geometry.centroid.x
grid['y'] = grid.geometry.centroid.y

# Create 3D plot
fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(grid['x'], grid['y'], grid['point_density'],
                     c=grid['point_density'], cmap='viridis',
                     s=50, alpha=0.7)
```

```
ax.set_xlabel('X Coordinate')
ax.set_ylabel('Y Coordinate')
ax.set_zlabel('Point Density')
plt.colorbar(scatter, label='Point Density')
plt.title("3D Visualization of Point Density")
plt.show()
```

The examples above demonstrate how to use GeoPandas for grid-based spatial analysis, which helps avoid the biases introduced by administrative boundaries. This approach allows for:

1. **Standardized Analysis Units**: Each grid cell has the same size and shape, unlike irregularly shaped administrative boundaries
2. **Finer Spatial Resolution**: Grid cells can reveal patterns that would be averaged out in larger administrative units
3. **More Objective Statistical Comparisons**: Calculations like density are directly comparable across the grid

The hands-on exercise I've provided gives students a complete workflow that:

1. Creates a regular grid over a study area
2. Performs point-in-polygon analysis to count features in each cell
3. Calculates standardized metrics like point density
4. Compares the grid-based approach with traditional administrative boundaries
5. Identifies statistical hot spots
6. Exports results for further analysis in QGIS

This comprehensive exercise demonstrates the practical advantages of grid-based analysis while teaching core GeoPandas skills, making it ideal for your 2-hour hands-on class.